

Student name: Lê Ngọc Anh Vũ

Student ID: 20236014

## LAB 7:

**Assignment 1: Create a project to implement Home Assignment 1. Compile and simulate it. Change the program parameters (register a0) and observe the execution results. Run the program in the single-step mode and pay attention to the changes in registers, especially the pc and ra registers.**

We change the value of program parameters (register a0) to 41 (a positive number) as the picture below:

```
# Laboratory Exercise 7 Assignment 1
.text
main:
    li a0, 41      # load input parameter
    jal abs        # jump and link to abs procedure

    li a7, 10      # terminate
    ecall

end_main:
# -----
# function abs
# param[in]   a0      the interger need to be gained the absolute value
# return      s0      absolute value
# -----
abs:
    sub s0, zero, a0    # put -a0 in s0; in case a0 < 0
    blt a0, zero, done  # if a0<0 then done
    add s0, a0, zero    # else put a0 in s0
done:
    jr ra
```

After executing the instruction “jal abs”, value of register ra changes from 0x00000000 to 0x00400008 which is the address of the next instruction “li a7, 10”. The register pc jumps from address 0x00400004 to address 0x00400010

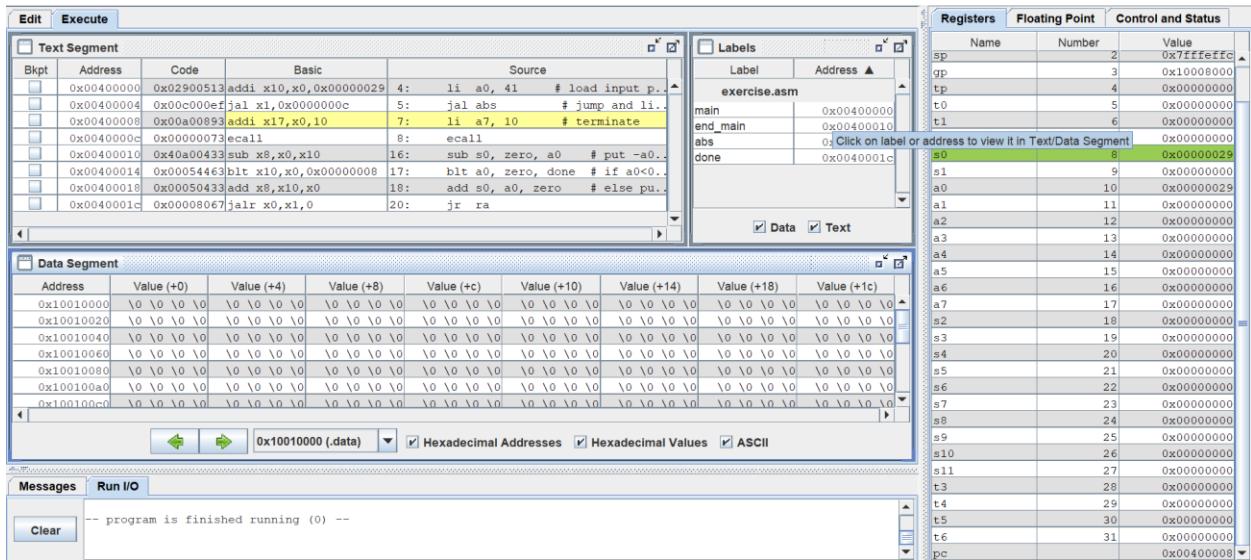
Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000029
a1	11	0x00000000

Name	Number	Value
zero	0	0x00000000
ra	1	0x00400008
sp	2	0x7ffffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000029
a1	11	0x00000000

pc 0x00400004

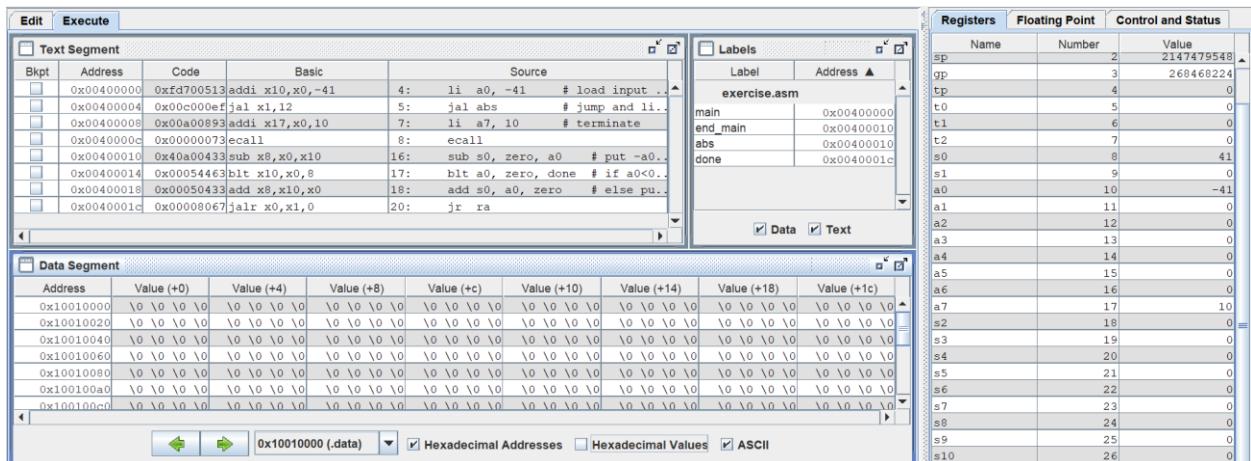
pc 0x00400010

After executing the instruction "jr ra", the register jumps again to the address 0x00400008



Then, the program ends with correct result which is s0=41 (the absolute value of 41)

Similarly to a negative value of program parameter, set a0 to -41 and the outcome is s0= 41, too.



**Assignment 2: Create a project to implement Home Assignment 2. Compile and simulate it. Change the program parameters (registers a0, a1, a2) and observe the execution results. Run the program in the single-step mode and pay attention to the changes in registers, especially the pc and ra registers.**

We change the value of program parameters a0, a1, a2 respectively to 1, 4, 5 as the picture below:

Edit Execute

exercise.asm\*

```

1 # Laboratory Exercise 7, Home Assignment 2
2 .text
3 main:
4     li    a0, 1      # load test input
5     li    a1, 4
6     li    a2, 5
7     jal   max       # call max procedure
8
9     li    a7, 10     # terminate
10    ecall
11 end_main:
12
13 #
14 # Procedure max: find the largest of three integers
15 # param[in] a0 integers
16 # param[in] a1 integers
17 # param[in] a2 integers
18 # return   s0   the largest value
19 #
20 max:
21     add   s0, a0, zero  # copy a0 in s0; largest so far
22     sub   t0, a1, s0    # compute a1 - s0

```

After executing the instruction “jal max”, value of register ra changes from 0x00000000 to 0x00400010 which is the address of the next instruction “li a7, 10”. The register pc jumps from address 0x0040000c to address 0x00400018

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00100513	addi x10,x0,1	4: li a0, 1 # load test input
	0x00400004	0x00400593	addi x11,x0,4	5: li a1, 4
	0x00400008	0x00500613	addi x12,x0,5	6: li a2, 5
	0x0040000c	0x00c000ef	jal x1,0x0000000c	7: jal max # call max procedur..
	0x00400010	0x00a00893	addi x17,x0,10	9: li a7, 10 # terminate
	0x00400014	0x00000073	ecall	10: ecall
	0x00400018	0x00050433	add x8,x10,x0	21: add s0, a0, zero # copy ..
	0x0040001c	0x408582b3	sub x5,x11,x8	22: sub t0, a1, s0 # compu..
	0x00400020	0x0002c463	bit x5,x0,0x00000008	23: blt t0, zero, okay # if al..

Registers Floating Point Control and Status

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffefffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000001
a1	11	0x00000004

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00100513	addi x10,x0,1	4: li a0, 1 # load test input
	0x00400004	0x00400593	addi x11,x0,4	5: li a1, 4
	0x00400008	0x00500613	addi x12,x0,5	6: li a2, 5
	0x0040000c	0x00c000ef	jal x1,0x0000000c	7: jal max # call max procedur..
	0x00400010	0x00a00893	addi x17,x0,10	9: li a7, 10 # terminate
	0x00400014	0x00000073	ecall	10: ecall
	0x00400018	0x00050433	add x8,x10,x0	21: add s0, a0, zero # copy ..
	0x0040001c	0x408582b3	sub x5,x11,x8	22: sub t0, a1, s0 # compu..
	0x00400020	0x0002c463	bit x5,x0,0x00000008	23: blt t0, zero, okay # if al..

Registers Floating Point Control and Status

Name	Number	Value
zero	0	0x00000000
ra	1	0x00400010
sp	2	0x7ffffefffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000001
a1	11	0x00000004

pc 0x0040000c

pc 0x00400018

Because a1 (=4) - s0 (=1) > 0, then s0 is set to a1 or s0 = 4.

Name	Number	Value
zero	0	0
ra	1	4194320
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	3
t1	6	0
t2	7	0
s0	8	4
s1	9	0
a0	10	1
a1	11	4

Again,  $a_2 (=5) - s_0 (=4) > 0$ , then  $s_0$  is set to  $a_2$  or  $s_0 = 5$

Name	Number	Value
zero	0	0
ra	1	4194320
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	1
t1	6	0
t2	7	0
s0	8	5
s1	9	0
a0	10	1
a1	11	4

After executing the instruction “jr ra”, the register pc jumps from the address 0x00400034 to the address stored in the register ra which is 0x00400010

pc	0x00400034
pc	0x00400010

Then, the program ends and the result which is the max value among  $a_0, a_1, a_2$  (1, 4, 5) is 5 stored in the register  $s_0$ . Correctly!!

Name	Number	Value
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	1
t1	6	0
t2	7	0
s0	8	5
s1	9	0
a0	10	1
a1	11	4
a2	12	5
a3	13	0

**Assignment 3: Create a project to implement Home Assignment 3. Compile and simulate it. Change the program parameters (registers  $s_0, s_1$ ), observe the process and results. Pay attention to changes in the  $sp$  register. Observe the memory pointed to by  $sp$  in the Data Segment window.**

We change the program parameters (register  $s_0, s_1$ ) that  $s_0 = 45, s_1 = 64$

exercise.asm

```

1 # Laboratory Exercise 7, Home Assignment 3
2 .text
3     li      $0, 45
4     li      $1, 64
5 push:
6     addi   sp, sp, -8      # adjust the stack pointer
7     sw      $0, 4(sp)      # push $0 to stack
8     sw      $1, 0(sp)      # push $1 to stack
9 work:
10    nop
11    nop
12    nop
13 pop:
14    lw      $0, 0(sp)      # pop from stack to $0
15    lw      $1, 4(sp)      # pop from stack to $1
16    addi   sp, sp, 8       # adjust the stack pointer

```

Initially, the register sp has value of 0x7ffffeffc. After executing the instruction “addi sp, sp, -8”, the register sp points to the address 0x7ffffeff4

The screenshot shows a debugger interface with three main panes:

- Text Segment:** Displays assembly code with columns for Bkpt, Address, Code, Basic, and Source. The source column highlights the instruction at address 0x00400000.
- Data Segment:** Displays memory dump tables for Address, Value (+0), Value (+4), Value (+8), Value (+c), Value (+10), Value (+14), Value (+18), and Value (+1c). It includes checkboxes for Hexadecimal Addresses, Hexadecimal Values, and ASCII.
- Registers:** Shows a table of registers with columns for Name, Number, and Value. The sp register is highlighted in yellow, showing its value as 0x7ffffeffc.

This screenshot shows the state of the debugger after the execution of the "addi sp, sp, -8" instruction at address 0x00400000.

- Text Segment:** The assembly code remains the same as in the previous screenshot.
- Data Segment:** The memory dump shows the initial values for addresses 0x7ffffef0 to 0x7fffff0c. The address 0x7ffffef0 is highlighted in yellow.
- Registers:** The sp register is highlighted in yellow, showing its new value of 0x7ffffeff4.

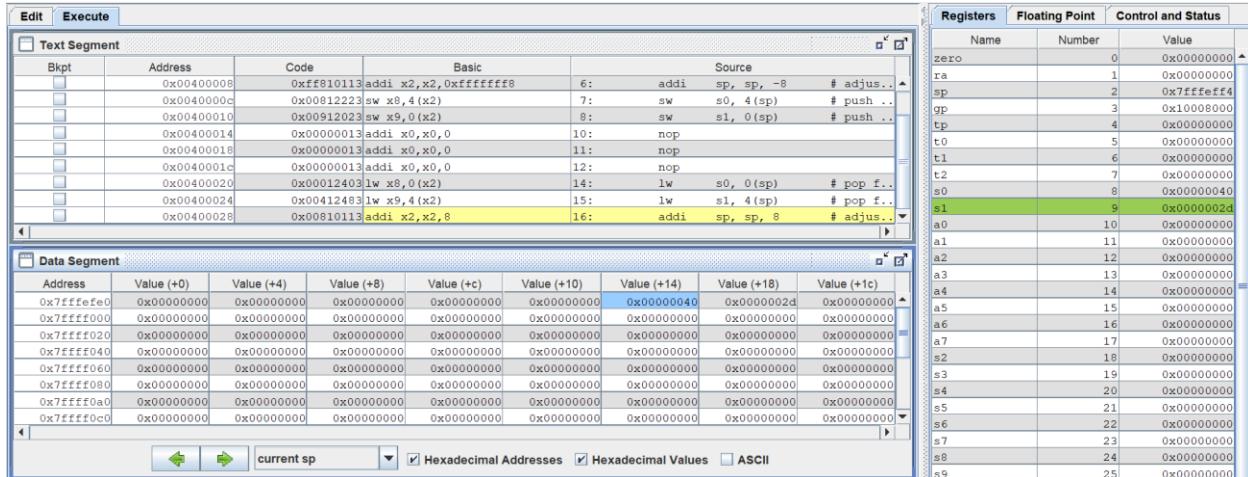
The value of s0, s1 is stored respectively in the address 0x7ffffeff8 ( sp + 4) and 0x7ffffeff4 (sp)

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeff4
gp	3	0x10000800
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x0000002d
s1	9	0x00000040
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000

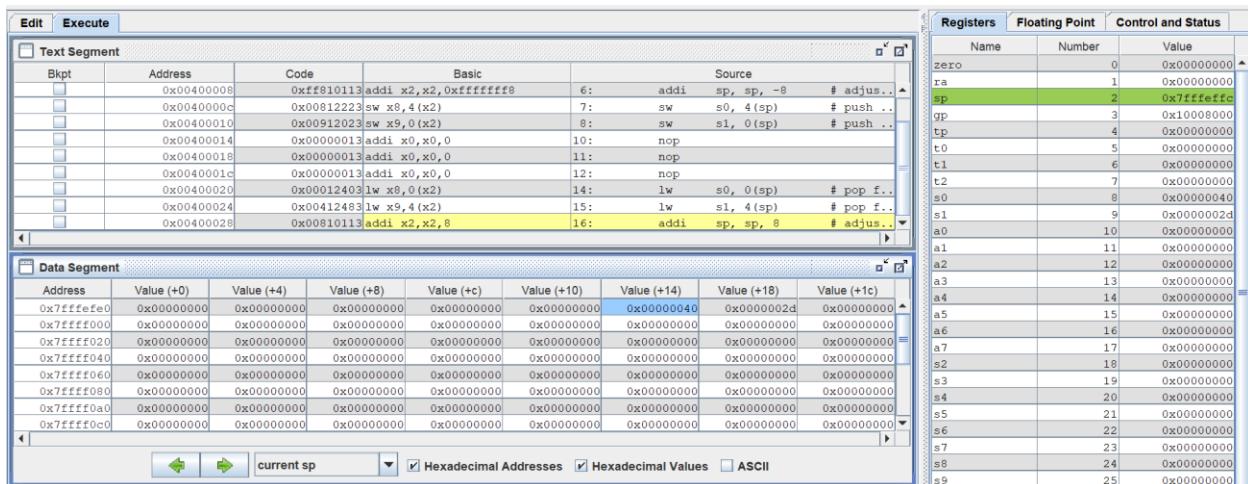
Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeff4
gp	3	0x10000800
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x0000002d
s1	9	0x00000040
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000

Then, loading the content in the address 0x7ffffeff4 (sp) to s0, the content in the address 0x7ffffeff8 (sp+4) to s1. At this time, s0 = 64, s1 = 45. They swapped the value.

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeff4
gp	3	0x10000800
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000040
s1	9	0x0000002d
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000



After all, we free space in the register sp, return the allocated stack memory by restoring the original value of the sp register. Then, the program ends.



**Assignment 4: Create a project to implement Home Assignment 4. Compile and simulate it. Change the parameter in the a0 register and check the result in the s0 register. Run the program in the single-step mode and observe the changes in the registers pc, ra, sp, a0, s0. List the values in the stack memory when executing the program with n = 3.**

We change the value of parameter in the register a0 to 4 as the picture below:

```

19
20 # -----
21 # Procedure WARP: assign value and call FACT
22 #
23 WARP:
24     addi    sp, sp, -4    # adjust stack pointer
25     sw      ra, 0(sp)    # save return address
26
27     li      a0, 4        # load test input N
28     jal     FACT         # call fact procedure
29
30     lw      ra, 0(sp)    # restore return address
31     addi    sp, sp, 4    # return stack pointer
32     jr      ra
33 wrap_end:
34

```

After executing the instruction “jal WARP”, the program jumps to label WARP and the register ra stores the address of the instruction right after “jal WARP” which is ra = 0x00400004. The register pc jumps to address of label WARP

The screenshot shows a debugger interface with three main panes: Text Segment, Data Segment, and Registers.

**Registers:**

Name	Number	Value
zero	0	0x00000000
ra	1	0x00400004
sp	2	0x7ffffeffc
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000

**Text Segment:**

Bkpt	Address	Code	Basic	Source
	0x00400000	0x020000ef	jal x1,0x00000020	7: jal WARP
	0x00400004	0x000405b3	addi x11,x8,x0	10: add a1, s0, zero # a0 = resu..
	0x00400008	0x03800893	addi x17,x0,0x00000038	11: li a7, 56
	0x0040000c	0x0fc10517	auipc x10,0x0000fc10	12: la a0, message
	0x00400010	0xffff450513	addi x10,x10,0xffffffff4	
	0x00400014	0x00000073	ecall	13: ecall
	0x00400018	0x00aa00893	addi x17,x0,10	16: li a7, 10 # terminate
	0x0040001c	0x00000073	ecall	17: ecall
	0x00400020	0xffffc10113	addi x2,x2,0xffffffffc	24: addi sp, sp, -4 # adjust stack ..

**Data Segment:**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x2074654b	0x20617571	0x686ee6974	0x61696720	0x68742069	0x6c206175	0x00203a61	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

**Messages:**

-- program is finished running (0) --

After that, the register sp is made a room for storing the address in register ra in order to mark the original place and be ready to execute the next subroutine. This address is stored in memory which the address is 0x7ffffeff8. This process is for program coming back to execute the next instruction below the subroutine. It's like a calling a function in high programming language.

The screenshot shows the assembly code and registers for a program. The assembly window displays the following code:

```

Text Segment
Bkpt Address Code Basic Source
0x00400008 0x03800893 addi x17,x0,0x00000038 11: li a7, 56
0x0040000c 0x0fc10517 auipc x10,0x0000fc10 12: la a0, message
0x00400010 0xffff450513 addi x10,x10,0xfffffffff4
0x00400014 0x00000073 ecall 13: ecall
0x00400018 0x00000893 addi x17,x0,10 16: li a7, 10 # terminate
0x0040001c 0x00000073 ecall 17: ecall
0x00400020 0xffffc10113 addi x2,x2,0xfffffffffc 24: addi sp, sp, -4 # adjust stack ...
0x00400024 0x00112023 sw x1,0(x2) 25: sw ra, 0(sp) # save return a...
0x00400028 0x00400513 addi x10,x0,4 27: li a0, 4 # load test imp...

```

The register window shows the following values:

Name	Number	Value
zero	0	0x00000000
ra	1	0x00400004
sp	2	0x7fffffe0
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000

After executing the instruction “ jal FACT”, the pc jumps from address 0x0040002c to address 0x0040003c. The register ra stores the address of the instruction right after “jal FACT” which is 0x00400030.

The screenshot shows the program counter (pc) at two different addresses:

- At address 0x0040002c, the pc is at 0x0040002c.
- At address 0x0040003c, the pc is at 0x0040003c.

The screenshot shows the assembly code and registers for a program. The assembly window displays the following code:

```

Text Segment
Bkpt Address Code Basic Source
0x0040001c 0x00000073 ecall 17: ecall
0x00400024 0xffffc10113 addi x2,x2,0xfffffffffc 24: addi sp, sp, -4 # adjust stack ...
0x00400028 0x00112023 sw x1,0(x2) 25: sw ra, 0(sp) # save return a...
0x0040002c 0x0040002c 0x010000ef jal x1,0x00000010 28: jal FACT # call fact pro...
0x00400030 0x000012083 lw x1,0(x2) 30: lw ra, 0(sp) # restore return ...
0x00400034 0x00410113 addi x2,x2,4 31: addi sp, sp, 4 # return stack ...
0x00400038 0x00000867 jalr x0,x1,0 32: jr ra ...
0x0040003c 0xffff810113 addi x2,x2,0xfffffffff8 41: addi sp, sp, -8 # allocate spac...

```

The register window shows the following values:

Name	Number	Value
zero	0	0x00000000
ra	1	0x00400030
sp	2	0x7fffffe0
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000004
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000

The stack is continually made space to store value of register ra and a0. It is important because this process helps the program could come back to the instruction right after the jump instruction called before, execute the rest instructions correctly. We have to store value of a0 in order to use a0 to compute without losing its original value during recursion subroutine takes place. This is because after each subroutine was done, we can load the initial value of a0 from memory in stack pointed by register sp.

The screenshot shows three identical copies of a debugger interface, likely from the QEMU debugger or a similar tool. Each copy displays a memory dump, assembly code, and register state.

**Registers:**

Name	Number	Value
zero	0	0x00000000
ra	1	0x04000030
sp	2	0x7fffffe0
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000004
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000

**Text Segment:**

Bkpt	Address	Code	Basic	Source
	0x00400034	0x00410113 addi x2,x2,4	31: addi sp, sp, 4 # return stack ..	
	0x00400038	0x00000067 jalr x0,x1,0	32: jr ra ..	
	0x0040003c	0xffff810113 addi x2,x2,0xffffffff8	41: addi sp, sp, -8 # allocate spac..	
	0x00400040	0x00112223 sw x1,4 (x2)	42: sw ra, 4(sp) # save ra regis..	
	0x00400044	0x00aa12023 sw x10,0 (x2)	43: sw a0, 0(sp) # save a0 regis..	
	0x00400048	0x00200293 addi x5,x0,2	45: li t0, 2	
	0x0040004c	0x00555663 bge x10,x5,0x0000000c	46: bge a0, t0, recursive	
	0x00400050	0x00100413 addi x8,x0,1	47: li s0, 1 # return the re..	
	0x00400054	0x0140006f jal x0,0x00000014	48: j done	

**Data Segment:**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7fffffe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00400004	0x00000000	0x00000000
0x7fffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffffa0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

**Registers:**

Name	Number	Value
zero	0	0x00000000
ra	1	0x04000030
sp	2	0x7fffffe0
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000004
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000

The recursion subroutine is called until  $a0 < 2$  which is the base case and  $s0$  storing the result is set to value of 1.

The screenshot shows the assembly code for the FACT subroutine in the Text Segment and the state of registers in the Registers window.

**Text Segment:**

Bkpt	Address	Code	Basic	Source
	0x0040003c	0xff810113	addi x2,x2,0xffffffff	41: addi sp, sp, -8 # allocate space..
	0x00400040	0x00112223	sw x1,4(x2)	42: sw ra, 4(sp) # save ra register..
	0x00400044	0x00a12023	sw x10,0(x2)	43: sw a0, 0(sp) # save a0 register..
	0x00400048	0x00202933	addi x5,x0,2	45: li t0, 2
	0x0040004c	0x00555663	bge x10,x5,0x0000000c	46: bge a0, t0, recursive..
	0x00400050	0x00100413	addi x8,x0,1	47: li s0, 1 # return the result..
	0x00400054	0x014000ef	jal x0,0x00000014	48: j done
	0x00400058	0xffff50513	addi x10,x10,0xffffffff	50: addi a0, a0, -1 # adjust index..
	0x0040005c	0xfefff0ef	jal x1,0xfffffffffe0	51: jal FACT # recursive..

**Registers:**

Name	Number	Value
zero	0	0x00000000
ra	1	0x04000060
sp	2	0x7ffffeffd8
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000001
s1	9	0x00000000
a0	10	0x00000001
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000

Then, the program jumps to label done, load the address in stack to register ra, which is the address of the instruction "lw s1, 0(sp) # load a0". And execute the part which is computing factorial step-by-step. The register pc jumps to address stored in ra

The screenshot shows the PC and RA registers. The PC contains the value 0x00400060, and the RA register contains the value 0x00400060.

pc	0x00400060
ra	1 0x00400060

This process has gone on until ra has value 0x00400030 and program exits the subroutine FACT

The screenshot shows the assembly code for the FACT subroutine in the Text Segment and the state of registers in the Registers window.

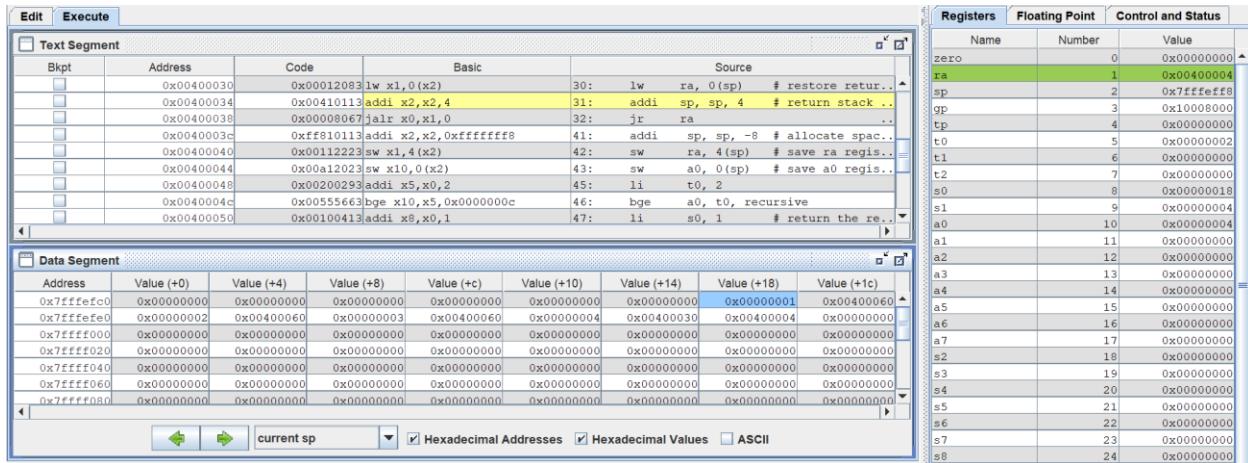
**Text Segment:**

Bkpt	Address	Code	Basic	Source
	0x00400054	0x014000ef	jal x0,0x00000014	48: j done
	0x00400058	0xffff50513	addi x10,x10,0xffffffff	50: addi a0, a0, -1 # adjust index..
	0x0040005c	0xfefff0ef	jal x1,0xfffffffffe0	51: jal FACT # recursive..
	0x00400060	0x00012483	lw x9,0(x2)	52: lw s1, 0(sp) # load a0
	0x00400064	0x02940433	mul x8,x8,x9	53: mul s0, s0, s1
	0x00400068	0x00412083	lw x1,4(x2)	55: lw ra, 4(sp) # restore r..
	0x0040006c	0x00012503	lw x10,0(x2)	56: lw a0, 0(sp) # restore a..
	0x00400070	0x00081013	addi x2,x2,8	57: addi sp,sp,8 # restore s..
	0x00400074	0x000008067	jalr x0,x1,0	58: jr ra # jump to c..

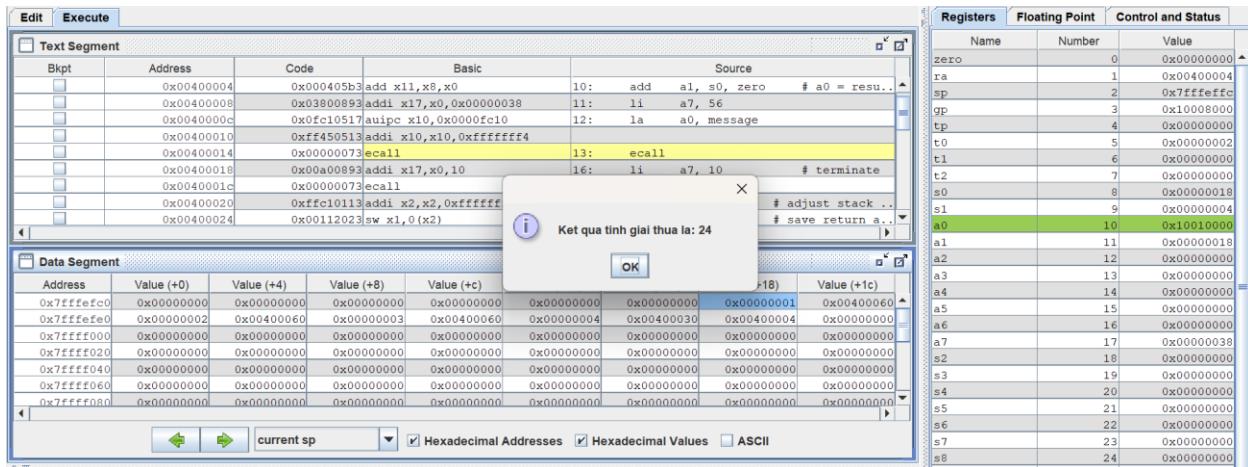
**Registers:**

Name	Number	Value
zero	0	0x00000000
ra	1	0x04000030
sp	2	0x7ffffeffd8
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000018
s1	9	0x00000004
a0	10	0x00000003
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000

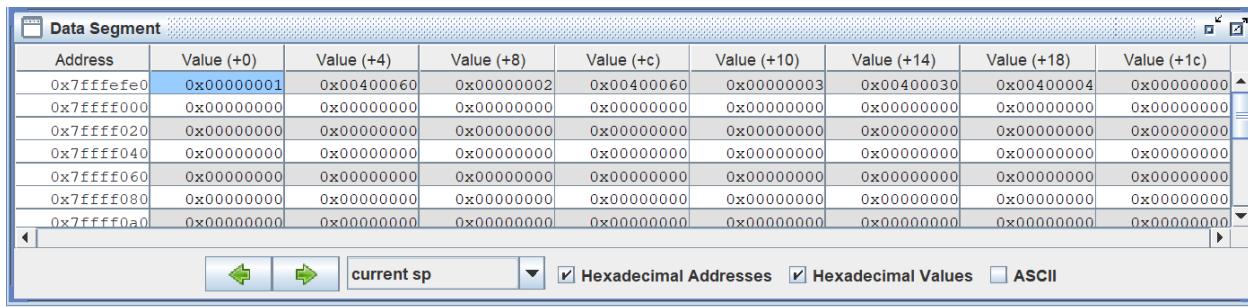
Then, register ra gets value of 0x00400004 and exits the label WARP. After each loading content from stack, sp is increased to remove space created in storing purpose.



After all, the program ends and register s0 =  $1*2*3*4$  or s0 = 24, print the output on dialog.



These are all values in stack with n=3:



**Assignment 5: Write a subroutine to find the largest value, the smallest value, and their respective positions in a list of 8 integers stored in the registers from a0 to a7. For example:**

- Largest: 9, 3 → The largest value is 9, stored in a3.
- Smallest: -3, 6 → The smallest value is -3, stored in a6.

Hint: Use the stack memory to pass parameters.

The source code is as below

lab 7.asm

```
1 # Laboratory Exercise 7, Assignment 5
2 .data
3     mess1: .asciz "The largest value is: "
4     mess2: .asciz "The smallest value is: "
5 .text
6 main:
7     add t0, sp, zero
8     li t3, 1
9     li a0, -3
10    li a1, 2
11    li a2, -6
12    li a3, 4
13    li a4, 10
14    li a5, 7
15    li a6, -1
16    li a7, 1
17    addi sp, sp, -28
18    sw a1, 0(sp)
19    sw a2, 4(sp)
20    sw a3, 8(sp)
21    sw a4, 12(sp)
22    sw a5, 16(sp)
```

lab 7.asm

```
22    sw a5, 16(sp)
23    sw a6, 20(sp)
24    sw a7, 24(sp)
25    li t1, 0
26    li t2, 0
27    add s0, a0, zero
28    add s1, a0, zero
29    jal find
30
31 print:
32     li a7, 4
33     la a0, mess1
34     ecall
35     li a7, 1
36     add a0, s1, zero
37     ecall
38     li a7, 11
39     li a0, 44
40     ecall
41     li a7, 11
42     li a0, 32
43     ecall
```

Edit Execute

lab 7.asm

```

43     ecall
44     li a7, 1
45     add a0, t2, zero
46     ecall
47     li a7, 11
48     li a0, 10
49     ecall
50     li a7, 4
51     la a0, mess2
52     ecall
53     li a7, 1
54     add a0, s0, zero
55     ecall
56     li a7, 11
57     li a0, 44
58     ecall
59     li a7, 11
60     li a0, 32
61     ecall
62     li a7, 1
63     add a0, t1, zero
64     ecall

```

Edit Execute

lab 7.asm

```

64     ecall
65     li a7, 11
66     li a0, 10
67     ecall
68 quit:
69     li a7, 10          # terminate
70     ecall
71 end_main:
72 find:
73     beq sp, t0, done
74     lw a1, 0(sp)
75     addi sp, sp 4
76     addi t3, t3, 1
77 min:
78     blt s0, a1, max
79     add s0, a1, zero
80     addi t1, t3, -1
81 max:
82     bgt s1, a1, find
83     add s1, a1, zero
84     addi t2, t3, -1
85     j find
86 done:
87     jr ra

```

The result is:

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7ffffe0	2	-6	4	10	7	-1	1	0
0x7ffff000	0	0	0	0	0	0	0	0
0x7ffff020	0	0	0	0	0	0	0	0
0x7ffff040	0	0	0	0	0	0	0	0
0x7ffff060	0	0	0	0	0	0	0	0
0x7ffff080	0	0	0	0	0	0	0	0
0x7ffff0a0	0	0	0	0	0	0	0	0

current sp

Hexadecimal Addresses  Hexadecimal Values  ASCII

Messages Run I/O

The largest value is: 10, 4  
The smallest value is: -6, 2

Clear